READ

```
{ READ   } [    ALL     ] [MULTI - FETCH - clause] [RECORD S] [IN] [FILE] view-name
{ BROWSE } [ (operand1) ]
         [PASSWORD = operand2]
         [CIPHER = operand3]
         [WITH REPOSITION]
         [sequence-range-specification]
         [STARTING WITH ISN = operand4]
         [WHERE logical condition]
         statement...
END-READ          (only in Structured Mode)
[LOOP]            (only in Reporting Mode)
```

Operand	Possible Structure					Possible Formats												Referencing Permitted	Dynamic Definition
Operand1	C	S					N	P	I		B							yes	no
Operand2	C	S				A												yes	no
Operand3	C	S					N											yes	no
Operand4	C	S					N	P	I		B							yes	no

Related Statement: FIND | HISTOGRAM

Function

The READ statement is used to read records from a database. The records can be retrieved in physical sequence, in Adabas ISN sequence, or in the value sequence of a descriptor (key) field.

This statement causes a processing loop to be initiated.

Number of Records - operand1/ALL

The number of records to be read may be limited by specifying *operand1* (enclosed in parentheses, immediately after the keyword READ) - either as a numeric constant (0 - 99999999) or as a variable, enclosed within parentheses, immediately after the keyword READ. For example:

```
   READ (5) IN EMPLOYEES ...

   MOVE 10 TO CNT(N2)
   READ (CNT) EMPLOYEES  ...
```

For this statement, the specified limit has priority over a limit set with a LIMIT statement.

If a smaller limit is set with the profile or session parameter LT, the LT limit applies.

To emphasize that *all* records are to be read, you can optionally specify the keyword ALL.

Notes:
If you wish to read a 4-digit number of records, specify it with a leading zero: (0*nnnn*); because Natural interprets every 4-digit number enclosed in parentheses as a line-number reference to a statement.
Operand1 is evaluated when the READ loop is entered. If the value of operand1 is modified within the READ loop, this does not affect the number of records read.

MULTI-FETCH Clause

Note:
This clause can only be used for Adabas databases.

[MULTI-FETCH ON | OFF | OF *value***]**

Note:
[MULTI-FETCH OF value] is not evaluated under Windows and UNIX. The default processing mode is applied.

In standard mode, Natural does not read multiple records with a single database call; it always operates in a one-record-per-fetch mode. This kind of operation is solid and stable, but can take some time if a large number of database records are being processed.

To improve the performance of those programs, Natural offers a new MULTI-FETCH clause, that allows one to read more than several records per database access.

For more information, see the section Multi-Fetch Clause in the Natural Programming Guide.

view-name

As *view-name*, you specify the name of a view, which must have been defined either within a DEFINE DATA statement or outside the program in a global or local data area.

In reporting mode, *view-name* may also be the name of a DDM.

PASSWORD and CIPHER Clauses

These clauses are applicable only to Adabas or VSAM databases. They cannot be used with Entire System Server.

The PASSWORD clause is used to provide a password when retrieving data from a file which is password-protected.

The CIPHER clause is used to provide a cipher key when retrieving data from a file which is enciphered.

See the statements FIND and PASSW for further information.

WITH REPOSITION

This option can only be applied if the underlying database is Adabas V7 (or above), Adabas 3.1.1 on Open Systems, VSAM or DL/I.

With a WITH REPOSITION option, you can make a READ statement sensitive for repositioning events. This allows you to reposition to another start value within an active READ loop. Processing of the READ statement then continues with the new start value.

A repositioning event is triggered by one of two ways when you use a READ statement with the WITH REPOSITION option:

1. When an ESCAPE TOP REPOSITION statement is executed.
   At execution of an ESCAPE TOP REPOSITION statement, Natural makes an instant branch to the loop begin and performs a restart; that is, the database repositions to a new record in the file according to the current content of the search value variable. At the same time, the loop-counter *COUNTER is reset to zero.
2. when a READ loop tries to fetch the next record from the database and the value of the system variable *COUNTER is "0".
   Note:
   If *COUNTER is set to "0" within the active READ loop, processing of the current record is continued; no instant branch to the loop begin is performed.
   You cannot trigger a reposition event in this fashion on Natural for Windows and for UNIX. This functionality has only been retained for compatibility reasons with Natural Version 3.1 for Mainframes. Therefore, it is not recommended that you use this process.

Functional Considerations

- If the READ statement has a loop-limit (e.g. READ (10) EMPLOYEES WITH REPOSITION ..) and a restart event was triggered, the loop gets another 10 new records, no matter how many records where already processed until the repositioning takes place.
- If an ESCAPE TOP REPOSITION statement is executed, but the innermost loop is not capable of repositioning (since the "WITH REPOSITION" keyword is not set in the READ statement or the posted loop statement is anything else but a READ), a corresponding runtime error is issued.
- Since the ESCAPE TOP statement does not allow a reference, you can only initiate a reposition event if the innermost processing loop is a READ ..WITH REPOSITION statement.
- A reposition event does not trigger the execution of the AT START OF DATA section nor, does it trigger the re-evaluation of the loop-limit operand (if it is a variable).
- If the search value was not altered, the loop repositions to the same record like at initial loop start.

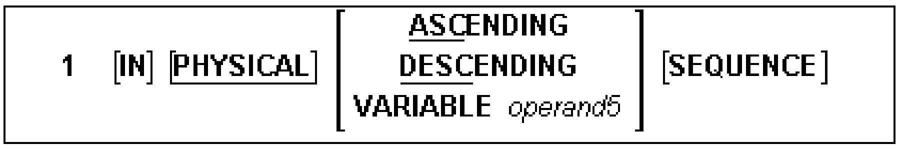
Example:

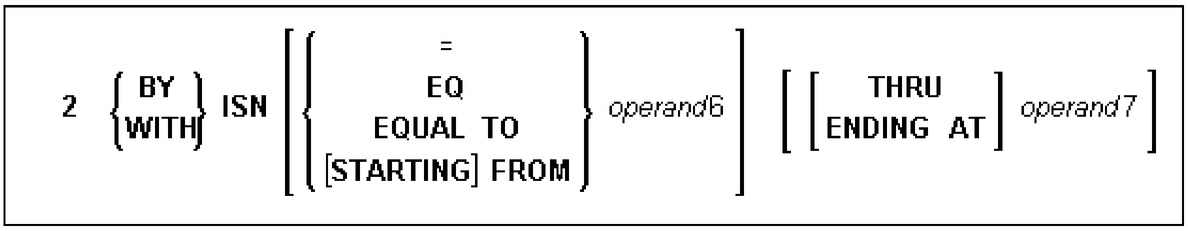
```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR     (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
  INPUT (IP=OFF AD=O) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN ESCAPE TOP REPOSITION
  END-IF
END-READ
...
```

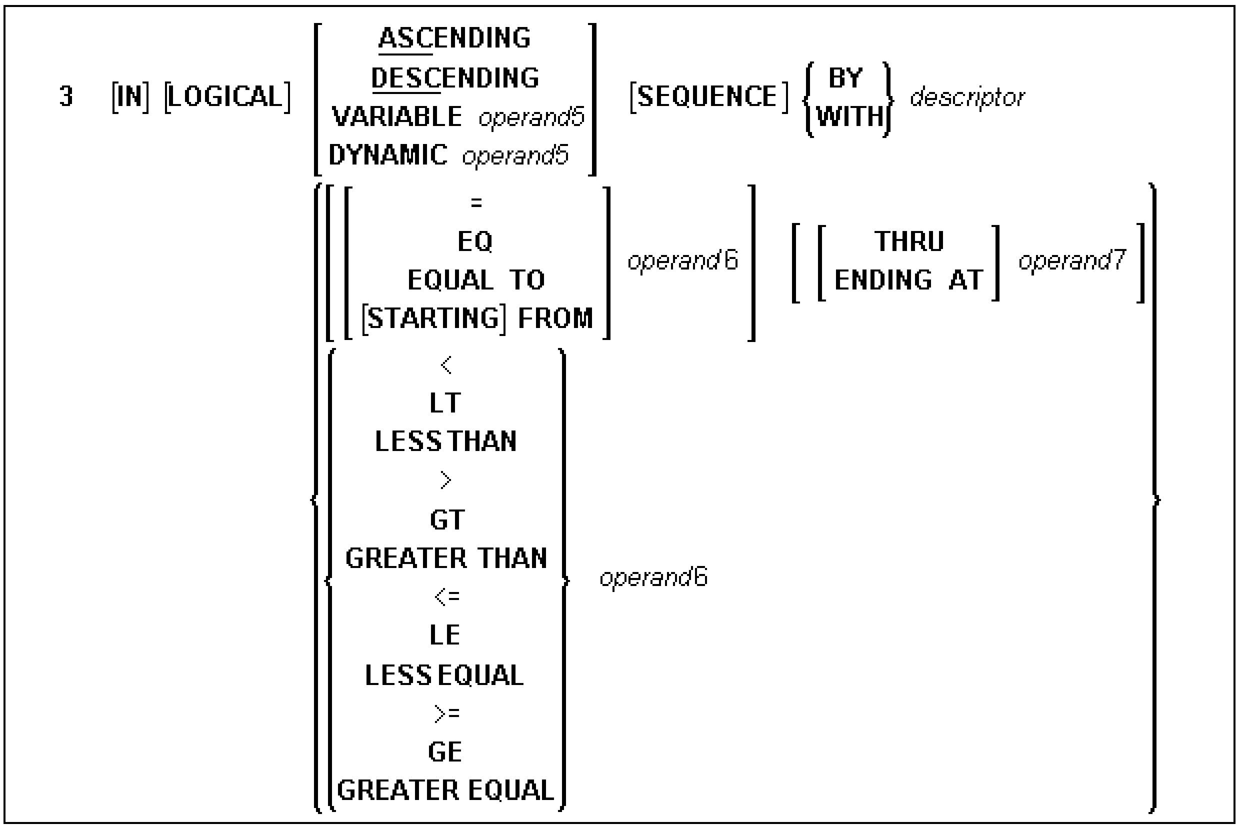
```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR     (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
  INPUT (IP=OFF AD=O) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN RESET *COUNTER
  END-IF
END-READ
...
```

sequence/range-specification

Note:
In Diagram 3 you will find comparators that may be used as of Natural Version 4.1.1 for Mainframes, Natural Version 6.1.1 for Windows/UNIX and above. If these comparators are used, the options ENDING AT, THRU and TO may not be used. These comparators are also valid for the HISTOGRAM statement.







Options [2] and [3] are not available with Entire System Server.

Operand	Possible Structure					Possible Formats												Referencing Permitted	Dynamic Definition
Operand5		S				A												yes	no
Operand6	C	S				A	N	P	I	F	B	D	T	L				yes	no
Operand7	C	S				A	N	P	I	F	B	D	T	L				yes	no

READ IN PHYSICAL SEQUENCE

PHYSICAL SEQUENCE is used to read records in the order in which they are physically stored in a database.

For VSAM databases, READ PHYSICAL can only be applied to ESDS and RRDS.

PHYSICAL is the default sequence.

READ BY ISN

READ BY ISN can only be used for Adabas and VSAM databases; for VSAM databases, it is only valid for ESDS and RRDS.

READ BY ISN is used to read records in the order of Adabas ISNs (internal sequence numbers) or VSAM RBAs (relative byte addresses of ESDS) or RRNs (relative record numbers of RRDS) respectively.

For XML databases, READ BY ISN is used to read XML objects according to the order of Tamino object IDs.

READ IN LOGICAL SEQUENCE

LOGICAL SEQUENCE is used to read records in the order of the values of a descriptor (key) field.

If you specify a descriptor, the records will be read in the value sequence of the descriptor. A descriptor, subdescriptor, superdescriptor or hyperdescriptor may be used for sequence control. A phonetic descriptor, a descriptor within a periodic group, or a superdescriptor which contains a periodic-group field cannot be used.

If you do not specify a descriptor, the default descriptor as specified in the DDM (field "Default Sequence") will be used.

For **DL/I databases**, the descriptor used must be either the sequence field of a root segment, or a secondary index field. If a secondary index field is specified, it must also be specified in the PROCSEQ parameter of a PCB. Natural uses this PCB and the corresponding hierarchical structure to process the database.
As HDAM databases use a randomizing routine to locate root segments, no sensible results will be returned when using READ LOGICAL for HDAM databases; therefore you should use READ PHYSICAL for HDAM databases.

For **VSAM databases**, LOGICAL is only valid for KSDS with primary and alternate keys defined and for ESDS with alternate keys defined.

If the descriptor used for sequence control is defined with null-value suppression (Adabas only), any record which contains a null value for the descriptor will not be read.

If the descriptor is a multiple-value field (Adabas only), the same record will be read multiple times depending on the number of values present.

Note:
READ IN LOGICAL SEQUENCE is also discussed in the Natural Programming Guide; see Statements for Database Access, READ Statement.

ASCENDING/DESCENDING/VARIABLE/DYNAMIC SEQUENCE

This clause only applies to Adabas, Adabas 3.1.1 on Open Systems, XML databases, VSAM and SQL databases. In a READ PHYSICAL statement, it can only be applied to VSAM and DB2 databases.

With this clause, you can determine whether the records are to be read in ascending sequence or in descending sequence.

- The default sequence is ascending (which may, but need not, be explicitly specified by using the keyword ASCENDING).
- If the records are to be read in descending sequence, you specify the keyword DESCENDING.
- If, instead of determining it in advance, you want to have the option of determining at runtime whether the records are to be read in ascending or descending sequence, you either specify the keyword VARIABLE or

DYNAMIC, followed by a variable (operand5). Operand5 has to be of format/length A1 and can contain the value "A" (for "ascending") or "D" (for "descending").

- If keyword VARIABLE is used, the reading direction (value of operand5) is evaluated at start of the READ processing loop and remains same until the loop is terminated, regardless if the operand5 field is altered in the READ loop or not.
- If keyword DYNAMIC is used, the reading direction (value of operand5) is evaluated before every record fetch in the READ processing loop and may be changed from record to record. This allows to change the scroll sequence from ascending to descending (and vice versa) at any place in the READ loop.

Note for Adabas databases:

- In order to use the sequences DESCENDING and VARIABLE, your system requires the following Adabas versions (or above): Version 3.1 on UNIX and Windows and Version 6.1 on mainframe computers.
- In order to use the DYNAMIC sequence, your system requires Adabas V7 (or above) and on mainframe computers and Adabas 3.1.1 on Open Systems.

Note for XML databases:
For XML databases, DYNAMIC SEQUENCE is not available.

Example of DESCENDING Option:

```
READ EMPLOYEES IN DESCENDING SEQUENCE BY NAME = 'SMITH'
```

Example of VARIABLE Option:

```
DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'>   /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
2 NAME
...
END-DEFINE
...
IF *PF-KEY = 'PF7'
THEN MOVE 'D' TO #DIRECTION
END-IF
READ #EMPVIEW IN VARIABLE #DIRECTION SEQUENCE BY NAME = 'SMITH'
...
END-READ
...
```

Example of DYNAMIC Option:

```
DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'>   /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
2 NAME
...
END-DEFINE
...
READ #EMPVIEW IN DYNAMIC #DIRECTION SEQUENCE BY NAME = 'SMITH'
   INPUT (AD=O) NAME
       / 'Press PF7 to scroll in DESCENDING sequence'
       / 'Press PF8 to scroll in ASCENDING  sequence'
   ..
   IF *PF-KEY = 'PF7' THEN MOVE 'D' TO #DIRECTION END-IF
   IF *PF-KEY = 'PF8' THEN MOVE 'A' TO #DIRECTION END-IF
END-READ
...
```

Further Examples:

See the programs READSCND and REAVSEQ in the library SYSEXRM.

STARTING FROM ... ENDING AT/TO

The STARTING FROM and ENDING AT clauses are used to limit reading to a set of records based on a user-specified range of values.

The STARTING FROM clause (= or EQ or EQUAL TO or [STARTING] FROM) determines the starting value for the read operation. If a starting value is specified, reading will begin with the value specified. If the starting value does not exist in the file, the next higher (or lower for a DESCENDING read) value will be used. If no higher (or lower for DESCENDING) value exists, the loop will not be entered.

In order to limit the records to an end-value, you may specify an ENDING AT clause with the terms THRU, ENDING AT or TO, that imply an inclusive range. Whenever the read descriptor field exceeds the end-value specified, an automatic loop termination is performed. Although the basic functionality of the TO, THRU and ENDING AT keywords looks quite similar, internally they differ in how they work.

THRU/ENDING AT

If THRU or ENDING AT is used, only the start-value is supplied to the database, but the end-value check is performed by the Natural runtime system, after the record is returned by the database. If the read direction is ASCENDING, you have to supply the lower value as the start-value and the higher-value as the end-value, since the start-value represents the value (and record) returned first in the READ loop. However, if you invoke a backwards read (DESCENDING), the higher value has to appear in the start-value and the lower-value in the end-value.

Internally, to determine the end of the range to be read, Natural reads one record beyond the end-value. If you have left the READ loop because the end-value has been reached, be aware that this last record is in fact not the last record within the demanded range, but the first record beyond that range (except if the file does not contain a further record after the last result record).

The THRU and ENDING AT clauses can be used for all databases which support the READ or HISTOGRAM statements.

TO

If the keyword 'TO' is used, both the start-value and the end-value are sent to the database and Natural does not perform checks for value ranges. If the end-value is exceeded, the database reacts the same as when "end-of-file" is reached and the database loop is exited. Since the complete range checking is done by the database, the lower-value (of the range) is always supplied in the start-value and the higher-value filled into the end-value, regardless if you are browsing in ASCENDING or DESCENDING order.

The TO option is only applicable if the underlying database is Adabas V7 (or above), Adabas 3.1.1 on Open Systems, Tamino, DB2, VSAM or DL1.

The following list describes the functional differences between the usage of the THRU/ENDING AT and TO options.

THRU/ENDING AT	TO
When the READ loop terminates because the end-value has been reached, the view contains the first record "out-of-range".	When the READ loop terminates because the end-value has been reached, the view contains the last record of the specified range.
If a end-value variable is modified during the READ loop, the new value will be used for end-value check on next record being read.	The end-value variable will only be evaluated at READ loop start. All further modifications during the READ loop have no effect.
An incorrect range (e.g. READ .. = ’B’ THRU ’A’) does not cause a database error, but just returns no record.	An incorrect range results in a database error (e.g. Adabas RC=61), because a value range must not be supplied in descending order.
If a READ .. DESCENDING is used with start- and end-value, the start value is used to position in the file, whereas the end-value is used by Natural to check for "end-of-range". Therefore the start-value is higher than (or equal to) the end-value.	Since both values are passed to the database, they have to appear in ascending order. In other words, the start-value is lower than (or equal to) the end-value, no matter if you are reading in ascending or descending order.
In order to check for range overflow, the descriptor value has to appear in the underlying database view; that is, it must be returned in the record buffer.	The descriptor is not required in the record fields returned.
The end-value check for an Adabas multi-value field (MU-field) or a sub-/super-/hyper-descriptor is not possible and leads to syntax error NAT0160 at program compilation.	You may specify an end-value for MU-fields and sub-/super-/hyper-descriptors.
Can be used for all databases.	Can only be used for Adabas V7 (or above), Adabas 3.1.1 on Open Systems, DB2, VSAM or DL1.

STARTING WITH ISN=operand4

This clause applies only to Adabas and VSAM databases.

Access to Adabas

This clause can be used in conjunction with a READ statement in physical or in logical (ascending/descending) sequence. The value supplied (operand4) represents an Adabas ISN (Internal Sequence Number) and is used to specify a definite record where to start the READ loop.

Logical Sequence

Even if documented with an equal character "=", the READ statement does not return only those records with exactly the start value in the corresponding descriptor field, but starts a logical browse in ascending or descending order, beginning with the start value supplied. If some records have the same contents in the descriptor field, they will be returned in an ISN-sorted sequence.

The STARTING WITH ISN clause is some kind of a "second level" selection criterion that applies only if the start value matches the descriptor value for the first record.

All records with a descriptor value that is the same as the start value and an ISN that is "less equal"("greater equal" for a descending READ) than the start ISN are ignored by Adabas. The first record returned in the READ loop is either

- the first record with descriptor = start value and an ISN "greater" ("less" for a descending READ) than the start ISN,
- or if such a record does not exist, the first record with a descriptor "greater" ("less" for a descending READ) than the start value.

Physical Sequence

The records are returned in the order in which they are physically stored. If a STARTING WITH ISN clause is specified, Adabas ignores all records until the record with the ISN that is the same as the start ISN is reached. The first record returned is the next record following the ISN=start ISN record.

Access to VSAM

This clause can only be used in physical sequence. The value supplied (operand4) represents a VSAM RBA (relative byte address of ESDS) or RRN (relative record number of RRDS), which is to be used as a start value for the read operation.

Examples

This clause may be used for repositioning within a READ loop whose processing has been interrupted, to easily determine the next record with which processing is to continue. This is particularly useful if the next record cannot be identified uniquely by any of its descriptor values. It can also be useful in a distributed client/server application where the reading of the records is performed by a server program while further processing of the records is performed by a client program, and the records are not processed all in one go, but in batches.

For an example, see the program REASISND in the library SYSEXRM.

WHERE Clause

```
WHERE logical-condition
```

The WHERE clause may be used to specify an additional selection criterion (*logical-condition*) which is evaluated *after* a value has been read and *before* any processing is performed on the value (including the AT BREAK evaluation).

The syntax for a *logical-condition* is described in the section Logical Condition Criteria.

If a LIMIT statement or a processing limit is specified in a READ statement containing a WHERE clause, records which are rejected as a result of the WHERE clause are not counted against the limit.

System Variables

The Natural system variables *ISN and *COUNTER are available with the READ statement.

The format/length of these system variables is P10. This format/length cannot be changed.

The usage of the system variables is illustrated below.

*ISN

The system variable *ISN contains the Adabas ISN of the record currently being processed.

For VSAM databases, *ISN contains either the RRN (for RRDS) or the RBA (for ESDS) of the current record.

For Tamino, *ISN contains the XML object ID.

For DL/I and SQL databases and with Entire System Server, *ISN is not available.

*COUNTER

The system variable *COUNTER contains the number of times the processing loop has been entered.

Example 1

```
/* EXAMPLE 'REAEX1S': READ (STRUCTURED MODE)
/**************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
LIMIT 3
/***************************************************
WRITE 'READ IN PHYSICAL SEQUENCE'
READ EMPLOY-VIEW IN PHYSICAL SEQUENCE
 DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/***************************************************
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOY-VIEW BY ISN
     STARTING FROM 1 ENDING AT 3
 DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/***************************************************
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOY-VIEW BY NAME
 DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/***************************************************
WRITE / 'READ IN NAME SEQUENCE STARTING FROM ''M'''
READ EMPLOY-VIEW BY NAME
     STARTING FROM 'M'
 DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
END-READ
/***************************************************
END
```

Equivalent reporting-mode example: See the program REAEX1R in the library SYSEXRM.

```
PERSONNEL          NAME           ISN       CNT
     ID
  --------- ------------------- --------- ---------

  READ IN PHYSICAL SEQUENCE
  50005600  MORENO                      2         1
  50005500  BLOND                       3         2
  50005300  MAIZIERE                    4         3

  READ IN ISN SEQUENCE
  50005800  ADAM                        1         1
  50005600  MORENO                      2         2
  50005500  BLOND                       3         3

  READ IN NAME SEQUENCE
  60008339  ABELLAN                   479         1
  30000231  ACHIESON                  884         2
  50005800  ADAM                        1         3

  READ IN NAME SEQUENCE STARTING FROM ’M’
  30008125  MACDONALD                 929         1
  20028700  MACKARNESS                780         2
  40000045  MADSEN                    509         3
```

Example 2 - Combining READ with FIND

The following program reads records from the EMPLOYEES file in logical sequential order based on the values of the descriptor NAME. A FIND statement is then issued to the VEHICLES file using the personnel number from the EMPLOYEES file as search criterion. The resulting report shows the name (read from the EMPLOYEES file) of each person read and the model of automobile (read from the VEHICLES file) owned by this person. Multiple lines with the same name are produced if the person owns more than one automobile.

```
/* EXAMPLE 'REAEX2': READ AND FIND
DEFINE DATA
   LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
LIMIT 10
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
     SUSPEND IDENTICAL SUPPRESS
FD.    FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
       IF NO RECORDS FOUND
          ENTER
       END-NOREC
       DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
               PERSONNEL-ID (RD.) FIRST-NAME (RD.)
               MAKE (FD.) (IS=OFF)
       END-FIND
     END-READ
END
```

```
 PERSONNEL      FIRST-NAME         MAKE
      ID
-------------- -------------- ---------------

20007500       VIRGINIA       CHRYSLER
20008400       MARSHA         CHRYSLER
                              CHRYSLER
20021100       ROBERT         GENERAL MOTORS
20000800       LILLY          FORD
                              MG
20001100       EDWARD         GENERAL MOTORS
20002000       MARTHA         GENERAL MOTORS
20003400       LAUREL         GENERAL MOTORS
30034045       KEVIN          DATSUN
30034233       GREGORY        FORD
11400319       MANFRED
```